In the Specification:

Please replace paragraph **[0004]** with new paragraph **[0004]**, shown below.

[0004] United States Patent Application No. ~~xx/xxx,xxx~~ 10/788,802, entitled "PROTECTION AGAINST INTERLEAVING TRANSACTIONS USING A TRANSACTION MANAGER," filed on February 27, 2004, ~~Attorney Docket No. BEAS1338US2,~~ currently pending.

Please replace paragraph **[0007]** with new paragraph **[0007]**, shown below.

[0007] The transaction manager may be provided by the application sever provided. One such provider of application servers is BEA Systems, Inc. of San Jose, California, who provide the Web Logic Server application server system. The WebLogic Server (WLS) Transaction Manager (TM) implements the J2EE™ JTA specification. This specification is based on the OpenGroup Distributed Transaction Processing Model (DTPM). A typical J2EE™ distributed transaction processing model 100 is depicted in Figure 1. Distributed Transaction Processing Model 100 includes application (App) 110, resource manager (RM) 120, and transaction manager (TM) 130. The TM coordinates two-phase commit (2PC) transactions that involve multiple resources. Resources developed by third parties may be utilized in WLS applications because they adhere to the J2EE™ standards. The App communicates with the RM using an API such as JDBC™ (for relational databases) and JMS (for queuing systems). The App controls transaction demarcation using the JTA API. The TM communicates with the RM during 2PC processing using the X/Open XA interface, specifically the XAResource interface as defined in the J2EE™ JTA specification. This interface provides methods for enlisting and delisting a resource in a global transaction, preparing the resource (first phase of 2PC), and committing or rolling back the resource (second phase of 2PC). There are also methods for use in failure recovery (recover), resource comparison

(isSameRM) and error processing (forget).

Please replace paragraph [0009] with new paragraph [0009], shown below.

[0009] Figure 2 illustrates a typical resource enlistment process 200. In process 200, the App 110 first begins a global transaction at step 210. The App then accesses the resource and invokes an update operation on the resource at step 220 using an API specific to the resource. For instance, the App may perform a JDBC™ update operation. When the update operation is invoked on the resource, the resource first makes a call into the TM at step 230 using the Transaction.enlistResource method. The resource passes the TM the XAResource object that the TM needs to utilize in order to perform the transaction enlistment and 2PC processing when the transaction is later committed or rolled back. In response to the enlistResource call, the TM will invoke XAResource.start on the resource at step 240. The application update is then performed in the resource at step 250 and is associated with the transaction that was specified during the enlistment start method. After the application request has been processed, the resource may invoke the delistResource method on the TM at step 260 to disassociate future operations from the transaction. The TM responds by calling XAResource.end on the resource at step 270. The process 200 of resource enlistment is then complete.

Please replace paragraph [0013] with new paragraph [0013], shown below.

[0013] FIGURE 1 is an illustration of a J2EE™ Distributed Transaction Processing Model in accordance with the prior art.

Please replace paragraph **[0018]** with new paragraph **[0018]**, shown below.

[0018] FIG. 3 illustrates a system 300 for implementing interleaving resource enlistment in accordance with one embodiment of the present invention. System 300 includes thread T1 320, App 321, TM 322, thread T2 330, App 331, resource connection object 340, and XAResource 341. In one embodiment, system 300 may be implemented on an application server system. The application server system may be on a single machine or several machines. Apps 321 and 331 may request a service from a resource during a connection involving different types of APIs. The system and methods of the present invention may be implemented using the different types of APIs and their related connection formats, including connections for JDBC™ and sessions for JMS.

Please replace paragraph **[0026]** with new paragraph **[0026]**, shown below.

[0026] The synchronization of the second thread 430 tasks with the tasks of first thread 420 is illustrated in FIG. 4. Once the TM 422 enlists the XAResource and obtains the lock to the resource at step 401, any attempted enlist from the second thread 430 is blocked. Thus, the enlist attempt at step 402 from thread 430 is blocked as it occurs later in time than the enlist step 401 of thread 420. In one embodiment, the lock may be implemented using Java™ monitor or some similar method. The lock is held until the thread has completed performing operations on the resource. After the lock on XAResource 442 is released at step 405, the second thread 430 may obtain the lock at step 406. At this point, the XAResource may begin performing work for the second thread. After the resource is delisted from the transaction ID at step 407, the XAResource work for the second thread ends at step 408 and the lock on the XAResource is released.